GUIT meeting 2014

Typesetting and highlighting Unicode source code with LaTeX: a package comparison

Roberto Giacomelli, Gianluca Pignalberi

Verona, 18 ottobre 2014

Definizione di "codice sorgente"

Nell'ambito informatico il **codice sorgente** è la descrizione di un algoritmo in un linguaggio di programmazione di livello più o meno alto, dunque un programma, e soddisfa queste due condizioni:

- è un testo memorizzato in un file digitale;
- è un testo che rispetta l'insieme delle regole sintattiche di un linguaggio informatico.

Gli elementi della sintassi come le **keyword** ed i **commenti** possono essere *evidenziati* con un diverso font e/o colore per aumentarne la leggibilità, per esempio...

Syntax highlighting di codice sorgente LaTeX

da così...

```
% dalla Gallery del sito GuIT...
% nel preambolo
\usepackage{guit}

% nel documento
\begin{itemize}
\item studiare la Guida \GuIT;
\item iscriversi al gruppo per;
 \begin{itemize}
   \item ricambiare l'aiuto ricevuto;
   \item aiutare a diffondere \TeX.
 \end{itemize}
\end{itemize}
```

...a così

```
% dalla Gallery del sito GuIT...
% nel preambolo
\usepackage{guit}

% nel documento
\begin{itemize}
\item studiare la Guida \GuIT;
\item iscriversi al gruppo per;
 \begin{itemize}
   \item ricambiare l'aiuto ricevuto;
   \item aiutare a diffondere \TeX.
 \end{itemize}
\end{itemize}
```

Comporre il codice nei nostri documenti

Di cosa parleremo in questo intervento?

poiché il codice sorgente UTF-8 è il formato di testo *obbligatorio* in moltissimi casi, è LaTeX in grado di comporlo correttamente?

- dell'inclusione del codice sorgente in documenti TeX con l'evidenziazione *automatica* della sintassi
- dell'utilizzo di codice sorgente in formato Unicode UTF-8
- dello studio sul campo dei principali pacchetti disponibili

La codifica del testo

La codifica del testo è la relazione biunivoca tra un *carattere* e un *numero*, ed è il ponte tra l'informazione digitale ed il testo

$$\underset{\text{quel che vede l'utente}}{\text{carattere}} \Leftrightarrow \underset{\text{quel che vede l'elaboratore}}{\text{numero}}$$

codifica $\rightarrow$ traduzione del testo in una sequenza numerica

decodifica $\rightarrow$ traduzione dell'informazione numerica nel testo

La codifica del testo: ASCII

La più importante codifica oggi esistente è ancora ASCII, *American Standard Code for Information Interchange*
Fu proposta nel 1961 dall'ingegnere dell'IBM **Bob Bemer**
Ecco alcuni codici ASCII a 7 bit tra i 128 disponibili (da 0 a 127):

carattere	$\Leftrightarrow$	numero
+	$\Leftrightarrow$	43
1	$\Leftrightarrow$	49
A	$\Leftrightarrow$	65
a	$\Leftrightarrow$	97

La codifica del testo: Unicode

Nel mondo sono state ideate molte altre codifiche per alfabeti di lingue diverse dall'inglese o specifiche di un sistema operativo
Da alcuni anni una sola codifica si è ormai imposta come standard per Internet, dalla posta elettronica alle pagine Web, e per lo scambio di informazioni tra sistemi diversi:

UNICODE

carattere $\Leftrightarrow$ code point

quel che vede l'utente — quel che vede l'elaboratore

La codifica UNICODE comprende decine di migliaia di simboli e alcune diverse modalità di rappresentazione chiamate *Unicode Trasformation Format*

Come funziona UNICODE

Al concetto base di biunivocità tra carattere e numero (*code point*) si aggiunge la trasformazione multi-byte perché occorre decidere come rappresentare il numero del code point in relazione all'unità base di memoria ad 8, 16 o 32 bit:

UTF-8: carattere $\Leftrightarrow$ code point $\Leftrightarrow$ byte (byte, (, byte)), byte)))

UTF-16: carattere $\Leftrightarrow$ code point $\Leftrightarrow$ double (, double)

UTF-32: carattere $\Leftrightarrow$ code point $\Leftrightarrow$ word

(a destra quel che vede l'elaboratore)

La più importante trasformazione fu ideata da **Rob Pike** e **Ken Thompson** nel 1992 per il sistema operativo Plan 9. Si tratta di

UTF-8

Come funziona UTF-8

Questa tabella rappresenta la trasformazione UTF-8 per convertire un code point in una sequenza da uno fino a quattro byte e viceversa

Byte 0	Byte 1	Byte 2	Byte 3
Range 1 → 0 … 7F			
0 a_6 a_5 a_4 a_3 a_2 a_1 a_0			
Range 2 → 80 … 7FF			
1 1 0 a_{10} a_9 a_8 a_7 a_6	1 0 a_5 a_4 a_3 a_2 a_1 a_0		
Range 3 → 800 … FFFF			
1 1 1 0 a_{15} a_{14} a_{13} a_{12}	1 0 a_{11} a_{10} a_9 a_8 a_7 a_6	1 0 a_5 a_4 a_3 a_2 a_1 a_0	
Range 4 → 10000 … 10FFFF			
1 1 1 1 0 a_{20} a_{19} a_{18}	1 0 a_{17} a_{16} a_{15} a_{14} a_{13} a_{12}	1 0 a_{11} a_{10} a_9 a_8 a_7 a_6	1 0 a_5 a_4 a_3 a_2 a_1 a_0

Il grande pregio della trasformazione UTF-8 è quello di risultare identico ad ASCII per i primi 128 simboli a livello di bit

L'importanza di UTF-8

La codifica testuale UTF-8 è importante perché:

- premette di usare direttamente l'alfabeto di una lingua che non usi caratteri latini (greco classico o cirillico, per esempio);
- permette lo scambio del testo in tutto il mondo e tra diversi sistemi operativi perché Unicode è uno standard internazionale;
- permette di preservare il più possibile nel futuro la lettura dei documenti grazie alla compatibilità con ASCII;
- permette di minimizzare la quantità di memoria necessaria per la codifica del testo.

Regola generale:

Codificare sempre il testo in UTF-8, quindi anche i sorgenti LaTeX

Caso studio: alcestis.tex

```
\documentclass[a4paper,12pt]{article}
\usepackage{eledmac}
\usepackage{xltxtra}
\usepackage{polyglossia}
\setmainlanguage{greek}
\setmainfont[Mapping=tex-text]{FreeSerif}
\sidenotemargin{left}
\begin{document}
\thispagestyle{empty}
\beginnumbering
\pstart
\noindent\edtext{}{\Afootnote{Mss.: \it VLPBΣD Con.:
\textit{Tournier, Pierson}.}}
\noindent Ἄδμηθ', ὁρᾷς γὰρ τἀμὰ πράγμαθ' ὡς
ἔχει,\ledsidenote{ ΑΛ.}\\
\pend
\endnumbering
\end{document}
```

Caso studio: primes.rs

```
fn primes_vec(n: uint) -> Vec<bool> {
    let lim = (n as f64).sqrt() as uint;
    let mut π_nums: Vec<bool> = Vec::from_elem(
        if n%2 == 0 {n/2 - 1} else {n/2}, // length
        true                              // value
    );

    for i in range(0, (lim-3)/2 + 1) {
        if *π_nums.get(i) {
            let π = 2*i + 3;        // prime value
            let mut k = (n/π-3)/2; // vec index
            while k+1 > i {
                *π_nums.get_mut((π*(2*k+3) - 3)/2) = false;
                k -= 1;
            }
        }
    }
    return π_nums;
}
```

I pacchetti considerati nel test comparativo

- classici
  - verbatim
  - fancyvrb
  - listings[1]
  - jvlisting
- pygments based
  - minted
  - verbments
  - PythonTEX

[1] Unico pacchetto classico capace di syntax highlighting

Le prove

Abbiamo effettuato 42 diversi test per controllare il syntax highlighting del codice, la corretta interpretazione di UTF-8 e l'interruzione di linea:

- due diversi file di codice sorgente UTF-8, uno in LaTeX ed uno in Rust (`http://www.rust-lang.org/`);
- tre diversi motori di composizione: pdfLaTeX, XeLaTeX e LuaLaTeX;
- sette diversi pacchetti.

Pacchetti classici

Tra i pacchetti *classici* che si basano solo sulle funzionalità di TEX solo listings è in grado di applicare il syntax highlighiting.

Purtroppo però listings non è compatibile con UTF-8: non interpreta correttamente i code point e li rappresenta malamente quando usato con pdfLATEX o li mette in posizioni errate se usato con XƎLATEX o LuaLATEX. I normali caratteri accentati non fanno, ovviamente, eccezione.

I pacchetti verbatim e fancyvrb possono comporre codice UTF-8 ma solo se utilizzati con i motori XƎLATEX o LuaLATEX assieme ai font che offrono i glifi corrispondenti.

Esempio 1

```
\documentclass[a4paper,12pt]{article
    }
\usepackage{eledmac}
\usepackage{xltxtra}
\usepackage{polyglossia}
\setmainlanguage{greek}
\setmainfont[Mapping=tex-text]{
    FreeSerif.otf}
\sidenotemargin{left}
\begin{document}
\thispagestyle{empty}
\beginnumbering
\pstart
\noindent\edtext{}{\Afootnote{Mss.:
    \it VB[U+FFFD] Con.:
    \textit{Tournier, Pierson}.}}
[U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD]
[U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD]
J+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD]
[U+FFFD][U+FFFD][U+FFFD],\edtext{}{\Afootnote{[U+FFFD]}\\
\pend
\endnumbering
\end{document}
```

Pacchetti basati su pygments

La libreria Pygments `http://pygments.org/` è scritta in Python come suggerisce il nome. Fornisce un programma chiamato `pygmentize`.

Il suo compito è quello di evidenziare il codice attraverso un parser specifico del linguaggio ed è quindi uno strumento del tutto generale.

Tra i formati di uscita di Pygments c'è anche LaTeX, così è possibile utilizzare `pygmentize` direttamente o attraverso specifici pacchetti come ad esempio minted.

Esempio 2

```
// Rust 0.11
// Euler's sieve implementation
// the bool vector holds only odd numbers
// start from 3 at index 0
fn primes_vec(n: uint) -> Vec<bool> {
    let lim = (n as f64).sqrt() as uint;
    let mut π _nums : Vec<bool> = Vec::from_elem(
        if n%2 == 0 {n/2 - 1} else {n/2}, // length
        true                              // value
    );

    for i in range(0, (lim-3)/2 + 1) {
        if *π _nums.get(i) {
            let π = 2*i + 3;          // prime value
            let mut k = (n/ π -3)/2; // vec index
            while k+1 > i {
                *π _nums.get_mut(( π *(2*k+3) - 3)/2) = false;
                k -= 1;
            }
        }
    }
    return π _nums;
}
```

Conclusioni

Anche se non esiste ancora il *pacchetto ideale*, gli utenti del sistema TEX possono includere codice sorgente UTF-8 con sintassi automaticamente evidenziata nei propri documenti ricordando che:

- il motore pdfTEX non è in grado di comporre correttamente un codice sorgente in UTF-8;
- occorre utilizzare motori di composizione compatibili con UTF-8 come XETEX o LuaTEX;
- occorre utilizzare font che offrono i glifi corrispondenti.

Ringraziamenti

Ringraziamo Claudio Beccari e Tommaso Gordini per la loro guida tematica sulle codifiche scaricabile dal sito del GUIT — `http://www.guitex.org/home/images/doc/GuideGuIT/introcodifiche.pdf` — e per aver risposto ad alcuni dei nostri quesiti sul tema...

...ed anche voi per l'attenzione

Domande?